29. September 2017 Studie · Mobile Sicherheit · 13 Min. Lesezeit

Verwendung von Policy-Enforcement Systemen zur Kontrolle von mobilen verteilten Systemen

Archiviert. Dieser Inhalt wird nicht mehr aktualisiert. Er bleibt zur Information verfügbar.

Stand 2017-09-20
Quelle https://technology2.a-sit.at/veroeffentlichungen/policy-enforcement-fuer-mobile-verteilte-systeme/

XACML als der de-facto Standard für Data Security Policies ermöglicht es, die Auswertung von Zugriffsberechtigungen zu zentralisieren und Regeln dafür an zentraler Stelle zu verwalten. In diesem Projekt wird XACML dahingehend erweitert, die Ausführung von verteilten mobilen Applikationen von zentraler Stelle aus kontrollieren zu können. Ohne Optimierungen würden Applikationen durch die Performance-Einbußen unbrauchbar werden. Besonders interessant macht dieses Verfahren außerdem, dass mobile Geräte nicht zwingend eine dauerhafte Verbindung zur Policy Infrastruktur benötigen.

1. Einleitung

Data Security Policies werden klassisch dazu verwendet, um im allgemeinen Fall die Zugangskontrolle von der eigentlichen Logik zu trennen. Die Autorisierung im Falle eines Webservices kann mittels Data Security Policies ausgegliedert werden. Auf abstrakter Ebene könnte hierfür ein Deployment wie in Abbildung 1 dargestellt verwendet werden.

© A-SIT, Partner für Sicherheitsfragen

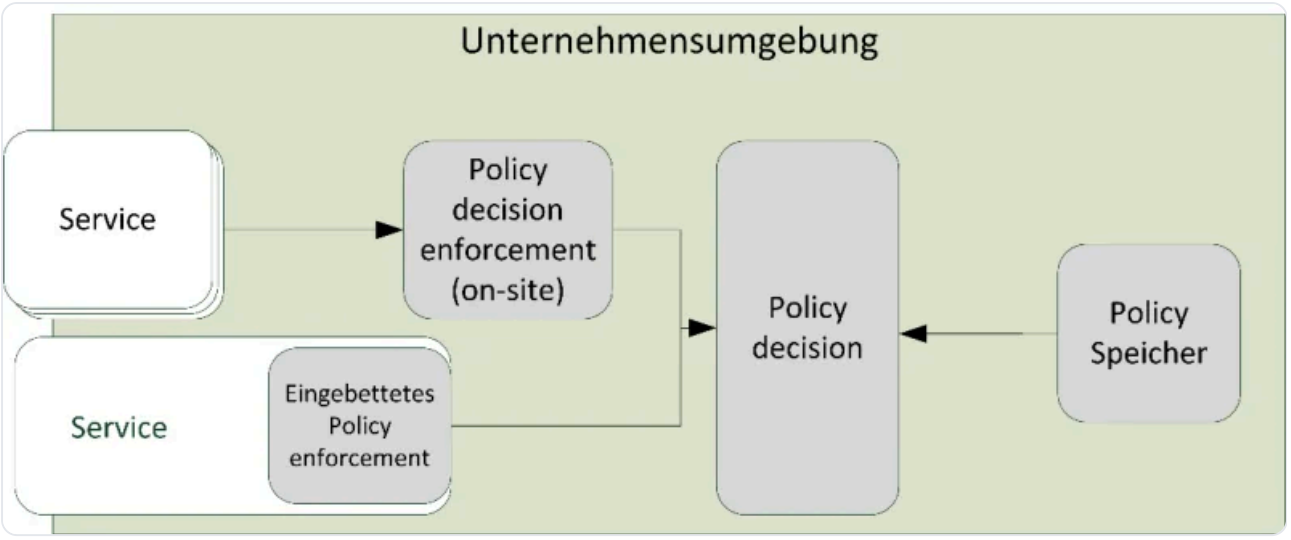


Abbildung 1: – High-Level Übersicht über das Deployment einer Policy Enforcement Infrastruktur Mehrere Services haben Zugangsbeschränkungen. Ein klassischer Ansatz hierzu wäre, konkrete Authentifizierungs- und Autorisierungsverfahren direkt in den Services zu verankern. In Unternehmensumgebungen mit einer Vielzahl von Services und deren zu verwaltenden Zugangsberechtigungen ist diese Strategie allerdings nicht zielführend. Die Abbildung zeigt einen vereinfachten Ansatz, um diese Autorisierungsdaten zentral zu verwalten. Hierbei ist die Policy decision Komponente die zentrale Komponente. Diese wird einerseits mit den definierten Policies aus dem Policyspeicher initialisiert und akzeptiert Autorisierungsanfragen in der Form: „Ist es Benutzer bzw. Benutzerin XY erlaubt auf Ressource X zuzugreifen?“. Im weiteren Verlauf liefert diese Komponente eine Aussage, ob die angeforderte Aktion erlaubt ist. Die Einhaltung dieser Aussage obliegt dann allerdings einer Enforcement Komponente. In klassischen Unternehmensdeployments sind hierbei zwei Strukturen vorherrschend.

- Services kommunizieren mit einer oder mehreren dedizierten *Enforcement* Komponenten, die als Gateways für die Kommunikation dienen bzw. die Aussage der *Policy Decision* Komponente in ein kompatibles Format übersetzen.
- Services werden bereits mit einer eingebetteten *Enforcement* Komponente ausgeliefert. Diese Komponente kommuniziert direkt mit der *Policy Decision* Komponente und erzwingt die Entscheidung. Das beschriebene Szenario ist ein Paradebeispiel für die Verwendung von *Policy Decision Languages* mit dem klaren Mehrwert, dass Zugriffsregeln an zentraler Stelle definiert werden können. Mit Blick auf die wachsende Zahl von mobilen Geräten und Applikationen, die zusehends dynamisch verteilt auf unterschiedlichen Geräten bzw. in der Cloud ausgeführt werden, drängt sich die Frage auf, ob diese Techniken auch zur Kontrolle von verteilten Applikationen im mobilen Kontext verwendet werden können.

2. Hintergrund

Extensible Access Control Markup Language (XACML) ist ein de-facto Standard im Bereich der *Policy Definition Languages* und wird vom OASIS Konsortium entwickelt und gewartet. XACML definiert einerseits eine *Policy Language,* um unterschiedlichste Sicherheitsanforderungen, Attributbasiert auszudrücken, andererseits werden Entitäten definiert, um die Integration bzw. Entwicklerinnen und Entwickler entsprechend anzuleiten. XACML basiert auf einem *Attribute-based Access Control (ABAC)* Ansatz. Dabei werden bereitgestellte Attributwerte mittels von XACML zur Verfügung gestellten Funktionen verglichen, um schlussendlich auf eine finale Zugriffsentscheidung zu kommen. Die eigentliche Definition von *Policies*

ist auf mehrere Ebenen aufgeteilt, um ein hohes Maß an Wiederverwendbarkeit zu erlangen. *Rules* beispielsweise als feinste Einheit in XACML können in mehreren *Policies* verlinkt werden. Diese wiederum können in *PolicySets*, die gröbste Einheit, zusammengefasst werden.

Eine beispielhafte *Policy* Definition wird im Folgenden illustriert:

```
4.    <AnyOf>
5.     <AllOf>
6.      <Match MatchId="...:function:string-equal">
7.       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">ExampleService</At
    tributeValue>
8.       <AttributeDesignator
9.        AttributeId="...:attribute:id"
10.       Category="...:attribute-category:service"
11.       DataType="http://www.w3.org/2001/XMLSchema#string"
12.       MustBePresent="true"/>
13.     </Match>
14.    </AllOf>
15.   </AnyOf>
16.  </Target>
17.  <Rule Effect="Deny" RuleId="DefaultDeny">
18.   <Target/>
19.  </Rule>
20.  <Rule Effect="Permit" RuleId="9to6">
21.   <Condition>
22.    <Apply FunctionId="...:function:and">
23.     <Apply FunctionId="...:function:time-greater-than">
24.      <Apply FunctionId="...:function:time-one-and-only">
25.       <AttributeDesignator
26.        AttributeId="...:environment:current-time"
27.        Category="...:attribute-category:environment"
28.        DataType="http://www.w3.org/2001/XMLSchema#time"
29.        MustBePresent="true"/>
30.      </Apply>
31.      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00Z</Attribute
    Value>
32.     </Apply>
33.     <Apply FunctionId="...:function:time-less-than">
34.      <Apply FunctionId="...:function:time-one-and-only">
35.       <AttributeDesignator
36.        AttributeId="...:environment:current-time"
37.        Category="...:attribute-category:environment"
38.        DataType="http://www.w3.org/2001/XMLSchema#time"
39.        MustBePresent="true"/>
40.      </Apply>
41.      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">18:00:00Z</Attribute
    Value>
42.     </Apply>
43.    </Apply>
44.   </Condition>
45.  </Rule>
46. </Policy>
```

Diese *Policy* betrifft ein Service, das über *ExampleService* identifiziert wird und definiert dafür zwei *Rules:*

- *DefaultDeny* stellt sicher, dass das Standardverhalten den Zugriff verwehrt.
- *9to6* erlaubt den Zugriff auf das Service zwischen 9:00 und 18:00 Uhr. Jeder *Rule* ist ein Effekt (*Effect*) zugeordnet, dessen Wirkung einsetzt, wenn die Bedingungen der *Rule* erfüllt sind. Im vorliegenden Beispiel wird die die *DefaultDeny Rule* immer zutreffen, zwischen 9:00 und 18:00 wird zusätzlich die zweite *Rule* zutreffen. In diesem Fall ergibt sich nun einerseits *Deny* aus der Ersten und

andererseits *Permit* aus der zweiten *Rule.* Um diese beiden Ergebnisse zu einem einzigen zusammenzufassen, ist in der übergeordneten Entität (*Policy*) ein *RuleCombiningAlgorithm* definiert. Dieser legt fest, wie unterschiedliche Ergebnisse kombiniert werden. Im vorliegenden Beispiel überschreibt *Permit* andere Resultate und hat somit Vorrang. Ein Ergebnis von *Deny* und *Permit* auf *Rule* Ebene wird also auf ein einzelnes *Permit* aufgelöst. Aus der Struktur der *Policy* kann bereits erahnt werden, dass durch den Attribut-basierten Ansatz die Grenzen sehr weit gesteckt sind und ein breites Spektrum der möglichen Modalitäten abgedeckt werden kann. Der zweite Teil der Spezifikation definiert Entitäten um die Auswertung und in weiterer Folge Einhaltung der *Policies* zu gewährleisten. Eine Übersicht über die definierten Komponenten und deren Zusammenspiel ist in Abbildung 2 ersichtlich:

- Der *Policy Administration Point (PAP)* verwaltet die definierten *Policies* und bietet Interfaces, um diese abzurufen bzw. um diese zu administrieren.
- Der *Policy Information Point (PIP)* dient als zusätzliche Quelle von Attributen im Auswertungsprozess, wie zum Beispiel zusätzliche Umgebungsparameter, Ressourcenparameter oder Attribute über die zugreifende Entität.
- Der *Policy Decision Point (PDP)* lädt die vorhandenen *Policies* und evaluiert Anfragen. Sollten im Zuge der Abarbeitung einzelne Attribute nicht verfügbar sein, werden diese über den PIP aufgelöst.
- Der *Policy Enforcement Point (PEP)* ist der Kontaktpunkt für Applikationen und sendet Anfragen an den PDP.

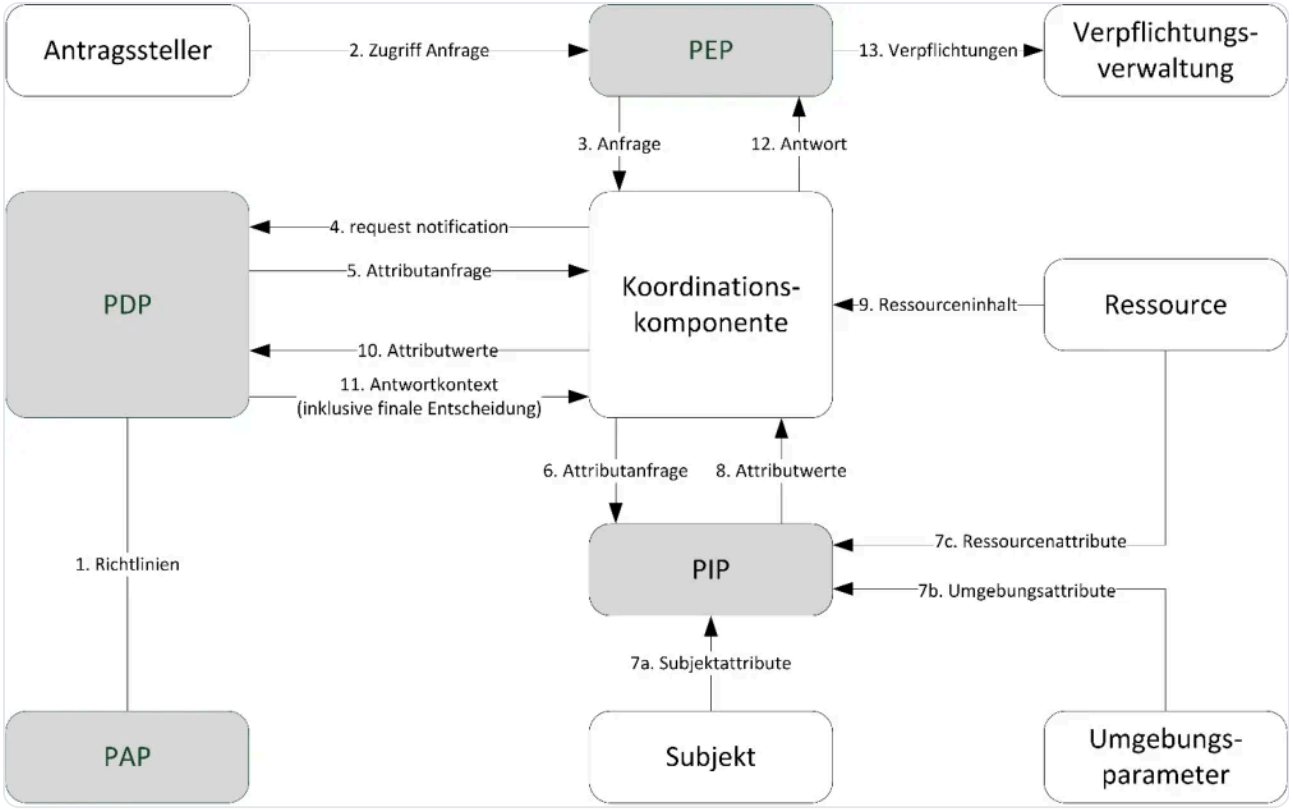


Abbildung 2: - Datenfluss bei XACML Die XACML Spezifikation an sich definiert nur Interfaces für die PDP Interaktion. Andere Interaktionen sind nur sehr lose definiert, ohne konkrete Technologien ins Spiel zu bringen. Trotzdem kann der allgemeine Ablauf der Evaluierung und Einhaltung von Policies wie in Abbildung 2 dargestellt im Folgenden erläutert werden. Die Annahme für diese Betrachtung ist, dass bereits alle Policies in den PAP geladen sind: (1) PDP lädt alle verfügbaren Policies bzw. hat darauf Zugriff. (2) Ein Service (mit oder ohne eingebetteten PEP) möchte auf eine geschützte Ressource zugreifen. Dieser Zugriff wird je nach Realisierung des PEP von diesem unterbrochen bzw. das Service sendet die Anfrage direkt an den PEP. (3) Der PEP sendet eine Zugriffsanfrage an die Koordinationskomponente. (4) Die Koordinationskomponente konvertiert die Anfrage in das vom PDP unterstützte Format und hat Zugriff zu den unterschiedlichen PIPs, um fehlende Attribute abzurufen. (5-10) Der PDP identifiziert noch fehlende Attribute, um die Policies auszuwerten und fordert diese von der Koordinationskomponente an. (11-12) Der PDP evaluiert die erhaltene Anfrage gegen die Policies und retourniert das erzielte Resultat an die Koordinationskomponente. Diese wiederum konvertiert das

Format zu einem applikations- bzw. servicespezifischen Format und sendet das Resultat zum PEP. (13) Der PEP kommt den mit diesem Resultat verknüpften Verpflichtungen nach. Verpflichtungen werden dazu verwendet, um den PEP Aktionen durchführen zu lassen, bevor ein Resultat als gültig angesehen werden kann. Damit ist die Abarbeitung einer einzelnen Abfrage beendet und der Service Workflow darf, je nach Ergebnis der Evaluierung, fortfahren. Der Ablauf zeigt bereits sehr deutlich die Komplexität dieses Ansatzes, trotzdem aber die Vorteile.

3. Anwendung in mobilen verteilten Umgebungen

Die Verwendung von mobilen Geräten wurde in den letzten Jahren auch für Ressourcenintensive Applikationen möglich. Trotzdem bedienen sich diese Applikationen unterschiedlicher Technologien um, wenn möglich, andere Rechenressourcen miteinzubeziehen. Ohne auf die Details der dabei verwendeten Technologie einzugehen kann zwischen zwei grundsätzlichen Ansätzen unterschieden werden:

- Mobile Geräte sind fix mit Cloud-Diensten verbunden, auf denen die benötigten Services laufen.
- Mobile Geräte suchen dynamisch nach potentiell verfügbarer Rechenleistung in ihrer Umgebung oder in der Cloud und verwenden diese Rechenleistung für ihre Berechnungen. Im ersteren Fall besteht bereits ein Vertrauensverhältnis zwischen den Cloud-Diensten und der Applikation, somit ist der Einsatz von *Policy Definition Languages* nur sehr beschränkt möglich. Im zweiten Fall allerdings kann zum Zeitpunkt des Deployments noch nicht abgeschätzt werden, welche Dienste von welchen Betreibern zur Laufzeit verfügbar sein werden und ob diese, von einer Vertrauensperspektive betrachtet, dafür geeignet sind. Im Folgenden wird ein Modell in zwei Ausführungen entwickelt um diesen Anforderungen nachzukommen. Die Betrachtungen verwenden die XACML Nomenklatur und können daher auch durch den sehr allgemeinen Aufbau auf andere Sprachen übertragen werden. Die Eckpunkte bzw. Anforderungen, die dieses Modell erfüllen muss, können wie folgt zusammengefasst werden:

(1) *Policies* müssen auf generische Weise festlegen können, welche Rechenressourcen vertrauenswürdig sind. (2) Abhängig von den Daten, die einer Funktion übergeben und damit potentiell verteilt ausgeführt werden, kann die erforderliche Vertrauensstellung beeinflusst werden. (3) Aus Performancegründen kann bereits abgeschätzt werden, dass nicht für jeden schützenswerten Funktionsaufruf in einer verteilten Applikation eine Auswertung der *Policies* angefordert werden kann. Ein Caching Mechanismus ist daher unerlässlich.

3.1. Modell

Ausgangspunkt für das Policy-Enforcement Modell für verteilte mobile Applikationen ist das Standard XACML Modell wie in Abbildung 3 dargestellt und bereits in den vorherigen Kapiteln besprochen.

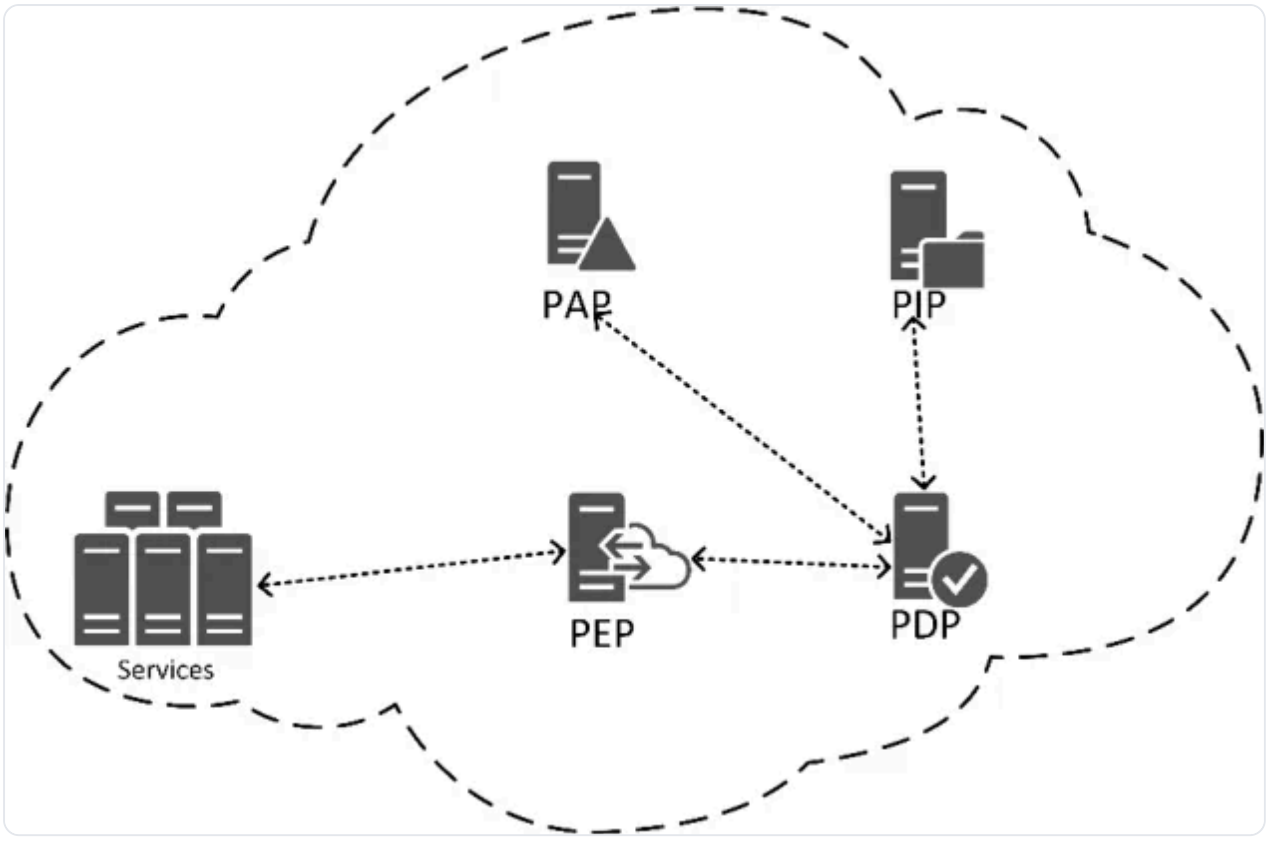


Abbildung 3: - XACML Enforcement Infrastruktur Dieses Modell soll nun für mobile Applikationen angepasst werden. Wie in der XACML Spezifikation vermerkt, ist die Anzahl der PEPs nicht beschränkt und möglicherweise sogar Instanz spezifisch. Bei der Ausführung von mobilen verteilten Applikationen hat das Gerät der Benutzerin bzw. des Benutzers, das kontrollierende mobile Gerät, die höchste Vertrauensstellung. Die Vertrauensstellung ist vergleichbar mit der der XACML Enforcement Infrastruktur im Ausgangsszenario. Dadurch können auch Komponenten der XACML Enforcement Infrastruktur auf das mobile Gerät verschoben werden. Ein erster Ansatz hierzu ist in Abbildung 4 dargestellt.

3.2. Vertrauenswürdige Ressourcen

Eine Frage die sich hierbei aufdrängt ist, wie Rechenressourcen klassifiziert in Policies aufgenommen werden können, die zur Laufzeit noch nicht, oder nur teilweise, bekannt sind. Dies

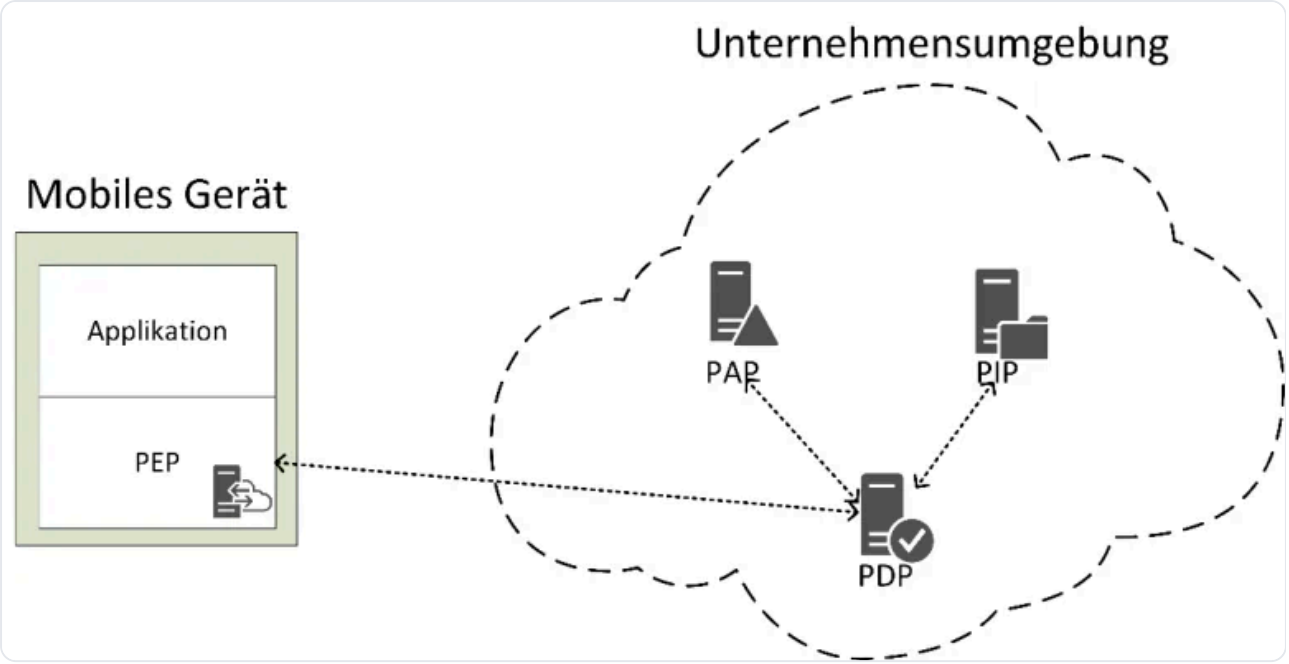


Abbildung 4: - Policy Enforcement für mobile verteilte Applikationen mit PEP am mobilen Gerät Die PEP Komponente wird von der Unternehmensumgebung auf das mobile Gerät verschoben und kommuniziert direkt mit der Policy Infrastruktur. Die Enforcement Komponente am mobilen Gerät muss Aufrufe der Applikation prüfen, die möglicherweise auf externen Rechenressourcen ausgeführt werden, und diese autorisieren. Damit kann eine zentrale Infrastruktur die Ausführung einer verteilten Applikation kontrollieren, ohne dass die Applikation direkt in der Unternehmensumgebung ausgeführt wird. In den folgenden Abschnitten werden die XACML Möglichkeiten gegen die definierten bzw. hergeleiteten Anforderungen abgewogen und gegebenenfalls Optimierungen entwickelt.

auch unter dem Gesichtspunkt, dass nicht alle Operationen zwingend eine vertrauenswürdige Rechenressource benötigen könnten. Für die Einbettung in XACML Policies können hierzu XACML *Obligations* verwendet werden. *Obligations* definieren, unter welchen Bedingungen ein vom PDP geliefertes Ergebnis Gültigkeit hat. Auf die Anfrage, ob eine Funktion verteilt ausgeführt werden darf, könnte also ein positives Ergebnis kommen, aber nur (in Form von *Obligations*) unter den Voraussetzungen, dass die Rechenressourcen die angegebenen Anforderungen erfüllen. Für zum Zeitpunkt des Deployments bereits bekannte Rechenressourcen können diese Anforderungen einfach durch Angabe eines Zertifikats, das zur Authentifizierung verwendet wird, realisiert werden. Ein Beispiel hierfür zeigt folgender Auflistung.

Herausfordernder ist allerdings der Fall, dass Rechenressourcen gänzlich unbekannt sind. Hierbei kann in der Policy auf *TRUSTED* Rechenressourcen verwiesen werden, wie in folgender Auflistung ersichtlich.

Die Vertrauenseinstufungsmethoden werden hierbei bereits zum Deploymentzeitpunkt festgelegt. Die eigentliche Einstufung erfolgt allerdings erst zur Laufzeit vom mobilen Gerät ausgehend. Auf die konkreten technischen Details der remote Attestation Methoden, die direkt vom mobilen Gerät je nach Verfügbarkeit ausgeführt werden können, wird hier nicht eingegangen. Im Überblick könnten aber unter anderem folgende Methoden verwendet werden:

- *Mobile-Device-Management (MDM) basierte Lösungen:* Hierbei definiert das Unternehmen

Regeln für mobile Geräte. Nur wenn diese erfüllt sind wird ein Gerät als Vertrauenswürdig eingestuft.

- *Google Safetynet basierte Lösungen* sind sehr ähnlich den MDM basierten Lösungen, jedoch

kann ein Unternehmen die Regeln, die erfüllt werden müssen, nicht selbst festlegen. Diese werden von Googles Dienst vorgegeben und sind nicht einsehbar. Deshalb wird Google Safetynet als externer Vertrauensdienstleister angesehen.

- *Intel SGX* bietet die Möglichkeit, Programmteile in geschützter Umgebung in sogenannten

Enclaves auszuführen. Diese besitzen zusätzlich remote Attestation Fähigkeiten, um zu beweisen, dass sie in vertrauenswürdiger Umgebung ausgeführt werden.

```
<ObligationExpression
 ObligationId="…:obligation:execution-mode"
 FulfillOn="Permit">
 <AttributeAssignmentExpression
  AttributeId="…:execution-mode">
  <AttributeValue
   DataType="http://www.w3.org/2001/XMLSchema#string">SPECIFIC</AttributeValue>
 </AttributeAssignmentExpression>
 <AttributeAssignmentExpression
  AttributeId="…:node">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
    {"endpoint": "client1", "identity": "[CERT-FINGERPRINT]"}
   </AttributeValue>
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
    {"endpoint": "client2", "identity": "[CERT-FINGERPRINT]"}
   </AttributeValue>
  </Apply>
 </AttributeAssignmentExpression>
</ObligationExpression>
```

Abbildung 5: - Vertrauensmodell

```
<ObligationExpression
 ObligationId="…:obligation:execution-mode"
 FulfillOn="Permit">
 <AttributeAssignmentExpression
  AttributeId="…:execution-mode">
  <AttributeValue
   DataType="http://www.w3.org/2001/XMLSchema#string">TRUSTED</AttributeValue>
 </AttributeAssignmentExpression>
</ObligationExpression>
```

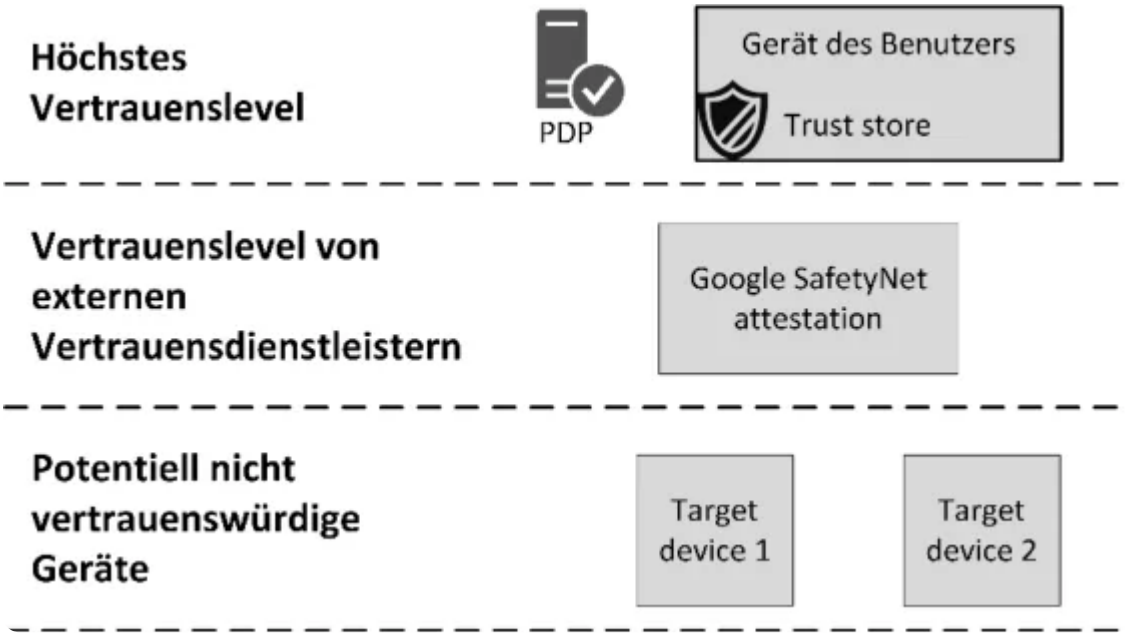
Dadurch ergibt sich ein Vertrauensmodell wie in Abbildung 5 dargestellt. Das Gerät des Benutzers, sowie die Policy Infrastruktur wird auf höchstem Vertrauenslevel eingestuft. Durch remote Attestation ist es möglich, andere Geräte auf dasselbe Vertrauenslevel anzuheben. Eine Sonderstellung haben hierbei externe Vertrauensdienstleister wie beispielsweise Google Safetynet. Hier muss mit Hilfe eines entsprechenden Risikomodells die Auswirkung von Datenverlust und die Garantien der Attestation Methode gegenübergestellt und abgewogen werden.

3.3. Policy Abgrenzung und Caching

Aufgerufene Funktionen können mit sicherheitskritischen Daten aufgerufen werden, aber auch mit unkritischen Daten. Policy Administratoren können Policies nicht nur an einzelnen Funktionen festmachen, sondern zusätzlich auch an deren Daten. Diese Möglichkeit ist Infrastrukturseitig durch den Attributbasierten Ansatz von XACML bereits vorgesehen. Dadurch ergibt sich allerdings die Situation, dass für jeden Funktionsaufruf, für den potentiell Policies definiert werden können, die Policy Infrastruktur

kontaktiert werden muss. Das würde zu erheblichen applikationsseitigen Performance Einbußen führen. Ein zusätzlicher clientseitiger Cachingmechanismus verhindert dieses Problem. Policy Administratoren können festlegen, unter welchen Umständen eine Policy Entscheidung für eine Funktion wiederverwendet werden darf. Darunter fallen die folgenden Möglichkeiten:

- Zeitbasierte Einschränkungen: Eine Entscheidung darf beispielsweise maximal eine Stunde wiederverwendet werden:
- Parameter basierte Einschränkungen: Eine Entscheidung darf nur wiederverwendet werden wenn Einschränkungen an die Aufrufparameter erfüllt sind:




```
<ObligationExpression
 ObligationId="…:obligation:validity"
 FulfillOn="Permit">
 <AttributeAssignmentExpression
  AttributeId="…:validity">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">3600</AttributeValue
>
 </AttributeAssignmentExpression>
</ObligationExpression>
```

```
<ObligationExpression
 ObligationId="…:obligation:validity"
 FulfillOn="Permit">
 <AttributeAssignmentExpression
  AttributeId="…:constraint">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
  {
  "op" : "and",
  "apply" : [
```

```
10.    {
11.     "op" : "test",
12.     "path" : "/parameter-values/arg1",
13.     "value" : "value1"
14.    },
15.    {
16.     "op" : "less",
17.     "path" : "/parameter-values/arg2",
18.     "value" : 5
19.    }]
20.   }
21.   </AttributeValue>
22.  </AttributeAssignmentExpression>
23. </ObligationExpression>
```

Dadurch können die Aufrufe zur Policy Infrastruktur minimiert werden und die Performanceauswirkungen in Grenzen gehalten werden.

3.4. Zusätzlicher PDP am mobilen Gerät

Die bisher beschriebenen Maßnahmen zur Optimierung der Performance trotz Einsatz der Policy Enforcement Infrastruktur können bei gegebenem Modell nur beschränkt weiter optimiert werden. Es werden in allen Fällen häufige Kommunikationen mit der Policy Infrastruktur notwendig sein. Eine weitere mögliche Optimierung ist das Deployment eines zusätzlichen PDP am mobilen Gerät, wie in Abbildung 6 dargestellt. Da sowohl das mobile Gerät des Benutzers sowie die Unternehmensumgebung auf derselben Vertrauensebene angesiedelt sind ergeben sich dadurch keine sicherheitsrelevanten Auswirkungen.

Abbildung 6 - Policy Enforcement für mobile verteilte Applikationen mit PEP und PDP am mobilen Gerät
Hierbei werden bei Initialisierung der Applikation alle relevanten Policies auf das mobile Gerät geladen. Entscheidungen die in weiterer Folge direkt am Gerät getroffen werden können erfordern keine Interaktion mit der Unternehmensumgebung. Das System kann also als zweistufige Policy Infrastruktur verstanden werden. Im Idealfall beschränkt sich die Interaktion zwischen mobilem Gerät und Policy Infrastruktur auf das Laden und regelmäßiges aktualisieren der Policies im Cache des mobilen Geräts. In praktischen Anwendungsszenarien werden Policies Attribute referenzieren die am mobilen Gerät nicht aufgelöst werden können und vereinzelte Interaktionen werden notwendig sein. Trotzdem kann dieses erweiterte Modell zur Optimierung der Interaktionen herangezogen werden.

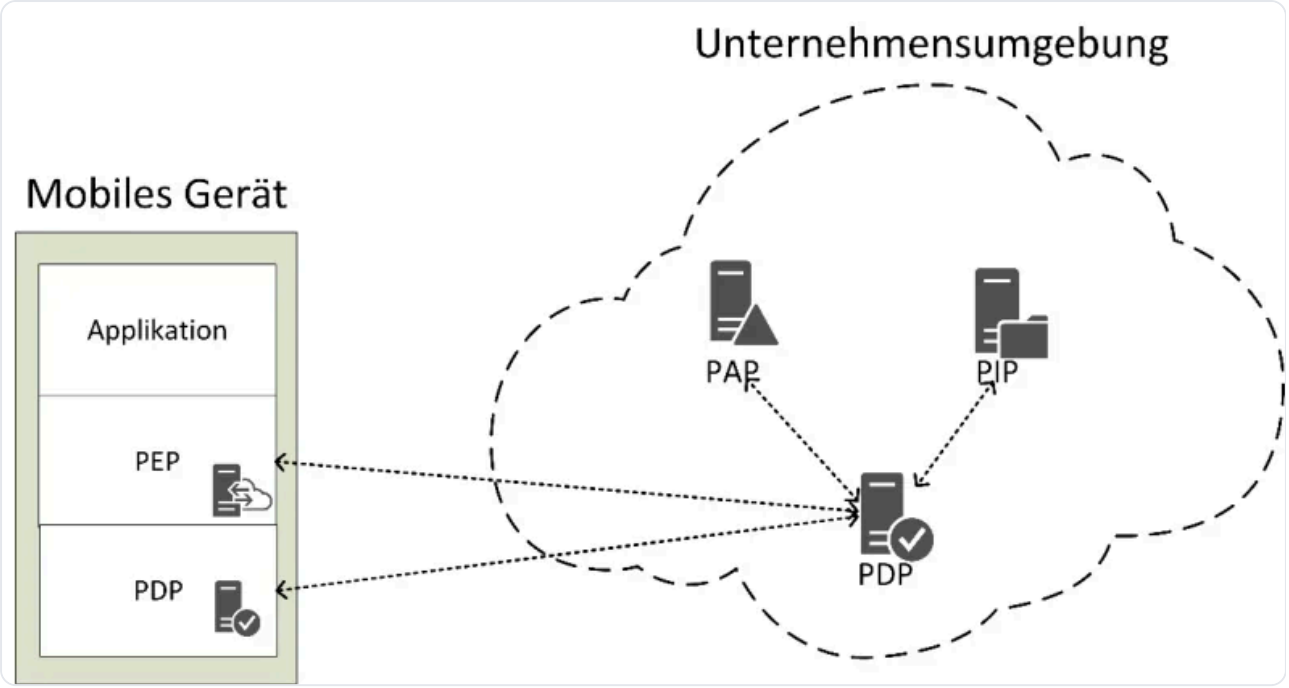


Abbildung 7: - Policy Infrastruktur Performance Es ist klar ersichtlich, dass die Evaluierungsdauer mit der Anzahl der potentiell zutreffenden Policies erheblich ansteigt. Obwohl die Evaluierungsdauer im Bereich von 100 – 1000 Policies gering ist, wäre diese bei direktem einbinden in Services deutlich spürbar.

4. Ergebnisse

Im Folgenden werden Ergebnisse aus den einzelnen Optimierungsschritten und deren Auswirkungen anhand des bereitgestellten Demonstrators gezeigt. Im ersten Schritt soll die Performance der Policy Infrastruktur verdeutlicht werden. Unter der Annahme, dass die Policy Infrastruktur mit 100, 1000, bzw. 10000 Policies bestückt wird, zeigt Abbildung 7 die durchschnittliche Dauer der Auswertung einer einzigen Anfrage, wobei *m* eine *General Purpose Computing* Instanz, und *c* eine *Data-processing Optimized* Instanz bei Amazon EC2 darstellt.

Unter Verwendung des beschriebenen Cachingverfahrens bzw. Optimierung ergibt sich ein gänzlich anderes Bild, wie in Abbildung 8 dargestellt. Hierbei gibt es nur beim erstmaligen Auswerten der Policy eine spürbare Verzögerung, danach beschränken sich die Performanceeinbußen sehr, je nach notwendigem Aufwand, um den Cache Eintrag auszuwerten.

5. Schlussfolgerungen

In diesem Projekt wurden die Möglichkeiten zur zentralen Kontrolle von dynamisch verteilten mobilen Applikationen untersucht. In klassischen, bereits bestehenden Systemen wird hierzu der de-facto Standard XACML eingesetzt. Dieses Projekt erreicht eine Adaptierung und Erweiterung für verteilte mobile Applikationen. Obwohl Policies zentral definiert werden können, benötigt ein mobiles Gerät nicht zwingend eine ständige Verbindung zur zentralen Policy Infrastruktur, sondern kann in weiten Teilen autonom agieren. In weiterer Folge sollte das Augenmerk auf den am mobilen Gerät eingebetteten PDP gelegt werden. Dieser bietet noch Potential um die lokalen Fähigkeiten zu verbessern und die notwendigen Interaktionen zu

minimieren. Außerdem ist denkbar, bei besonders sensiblen Applikationen nur ein Subset der Policies am mobilen Gerät auszuwerten und für die restlichen Policies die Auswertung in der Unternehmensumgebung zu erzwingen.

	100 policies	1000 policies	10000 policies
m	107ms	272ms	1912ms
c	88ms	212ms	1663ms

Abbildung 8: - Cache Performance Weitere Optimierungen beschränken die nicht vernachlässigbaren Performanceeinbußen im besten Fall auf ein initiales Laden beim Starten der Applikation.

	Android	Desktop
Ohne Parameter Einschränkungen	1,71ms	0,32ms
Mit Parameter Einschränkungen	3,95ms	1,43ms